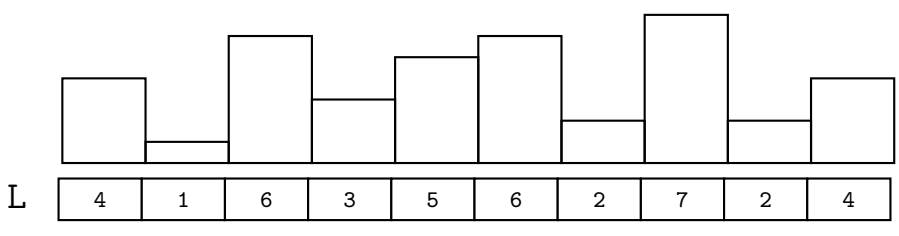
O problema do maior retângulo

Imagine que L é uma lista de tamanho n que contém números positivos

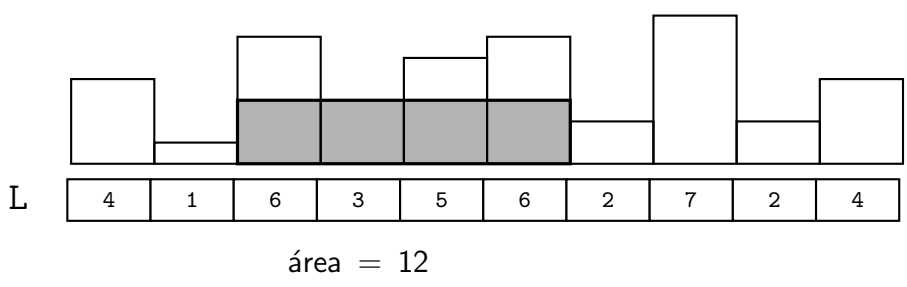
A ideia é que cada número corresponde à altura de uma coluna — (*de largura 1*)



E o problema é

→ *Encontrar um retângulo com a maior área possível*
que fica inteiramente no interior das colunas

No exemplo acima, isso poderia nos dar



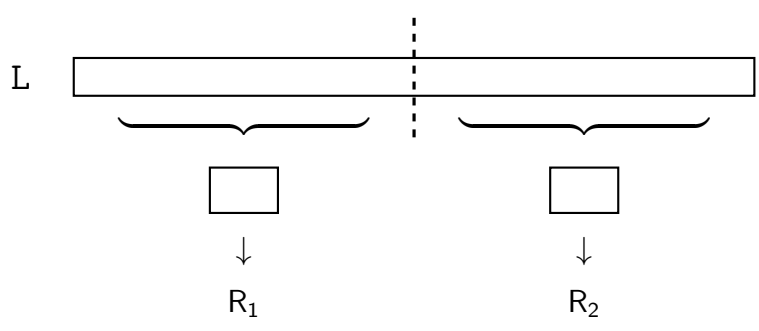
Certo.

É bem fácil resolver esse problema em tempo $O(n^2)$ — (*como?*)

Mas, $O(n^2)$ é uma porcaria ...

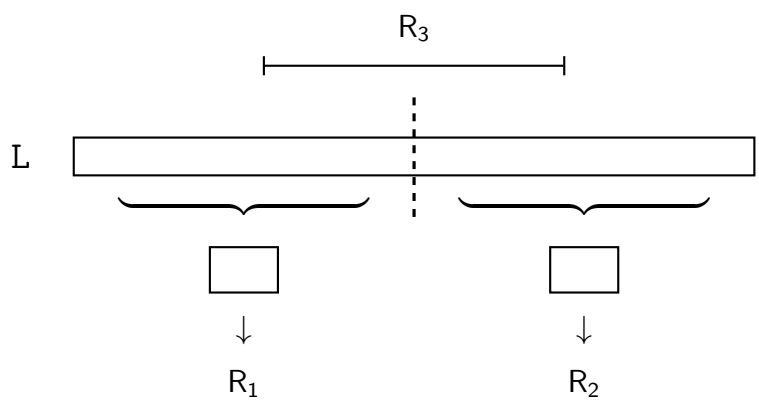
Então, a gente vai tentar algo melhor com a divisão e conquista.

Quer dizer, imagine que a gente quebra a lista no meio, e resolve os problemas de cada lado — (*recursivamente*)



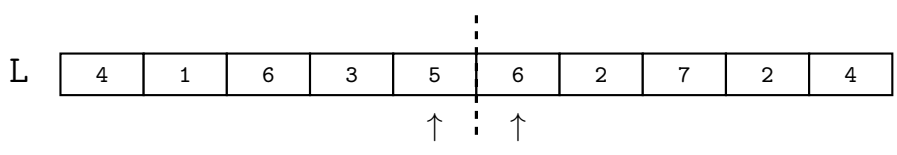
Isso, é claro, ainda não resolve o nosso problema.

Porque ainda é preciso verificar os retângulos que tem uma parte em cada metade

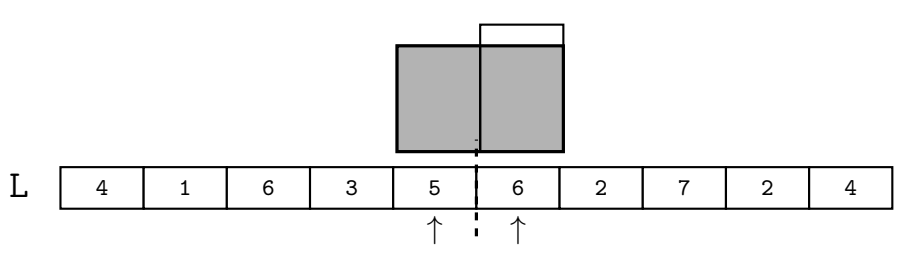


Então agora, a gente se concentra nesse problema.

E para isso, a gente foca a atenção nas duas colunas do meio



Quer dizer, o maior retângulo que cabe nessas colunas é dado pela altura da menor delas



Isso já nos dá uma primeira solução

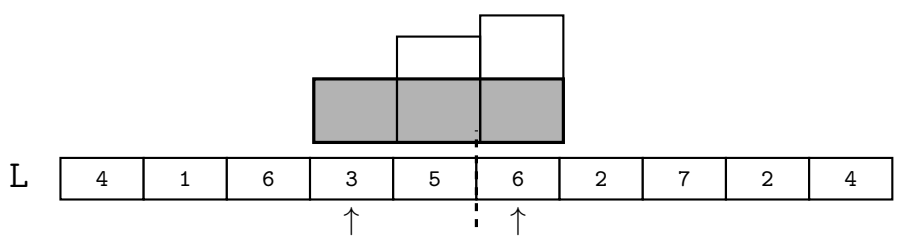
$$\texttt{R} \quad = \quad 2 * 5 \quad = \quad 10$$

E agora é preciso ver se é possível conseguir algo melhor do que isso

Para onde a gente vai?

Ora, a gente vai na direção da maior coluna.

E nesse caso, isso corresponde a ir para o lado esquerdo



Ao fazer isso, a altura diminui um pouquinho.

Mas nós ganhamos alguma coisa na largura.

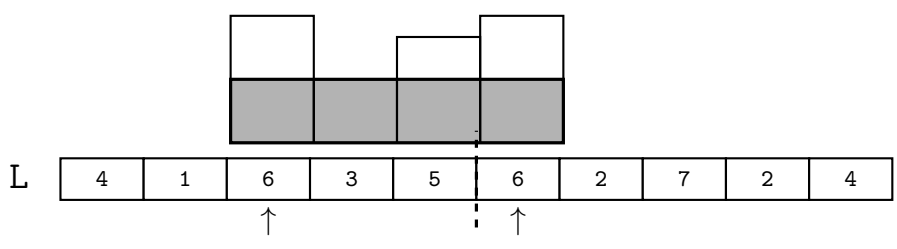
O resultado é

$$\texttt{R} \quad = \quad 3 * 5 \quad = \quad 9$$

o que ainda não melhora a nossa solução.

Então, a gente dá mais um passo.

E o passo é novamente para a esquerda



Dessa vez a altura não diminui — (*e ela não pode nunca aumentar — porque?*)

Mas nós ganhamos alguma coisa na altura.

O resultado é

$$\texttt{R} \quad = \quad 4 * 5 \quad = \quad 12$$

E dessa vez a gente melhorou a nossa solução.

Agora, é só continuar fazendo a mesma coisa.

E ver qual é a melhor solução que a gente consegue obter

A coisa fica assim

```
func retanguloMeio ( L, inicio, meio, fim )

    i ← meio;     j ← meio+1

    MR ← larg * alt

    larg ← 2;     alt ← Min { A[i],A[j] }

    Enquanto ( i ≥ inicio  OU  j ≤ fim )

        larg ++

        Se ( j > fim  OU  ( i ≥ inicio  E  L[i] > L[j] ) )

            alt ← Min { alt,L[i] }     i---

        Senão

            alt ← Min { alt,L[j] }     j++

        R ← larg * alt

        Se ( R > MR )     MR ← R

    Retorna ( MR )
```

É bem fácil ver que esse algoritmo executa em tempo $O(n)$.

Agora, nós podemos escrever o nosso algoritmo de divisão e conquista

```
T(n) =  func maiorRetangulo-DC ( L, inicio, fim )

          Se ( inicio = fim )     Retorna ( L[inicio] )
   O(1)
          meio ← ( inicio + fim ) / 2

T(n/2) ←  R1 ← maiorRetangulo-DC ( L, inicio, meio )

T(n/2) ←  R2 ← maiorRetangulo-DC ( L, meio+1, fim )

  O(n) ←  R3 ← retanguloMeio ( L, inicio, meio, fim )

          Retorna ( Max { R1, R2, R3 } )
```

Análise

Seguindo as anotações, nós obtemos a seguinte equação de recorrência para o tempo de execução do algoritmo

$$T(n) \quad = \quad 2 \cdot T\left(\frac{n}{2}\right) \quad + \quad O(n)$$

E todo mundo sabe que isso dá

$$T(n) \quad = \quad O(n \log n)$$

— (*ou será que não?*)